Sound Numerical Computations in Abstract Acceleration*

Dario Cattaruzza[1], Alessandro Abate[1], Peter Schrammel[2], and Daniel Kroening[1]

[1] Department of Computer Science, University of Oxford, UK
[2] School of Engineering and Informatics, University of Sussex, UK

Abstract. Soundness is a major objective for verification tools. Methods that use exact arithmetic or symbolic representations are often prohibitively slow and do not scale past small examples. We propose the use of numerical floating-point computations to improve performance combined with an interval analysis to ensure soundness in reach-set computations for numerical dynamical models. Since the interval analysis cannot provide exact answers we reason about over-approximations of the reachable sets that are guaranteed to contain the true solution of the problem. Our theory is implemented in a numerical algorithm for Abstract Acceleration in a tool called Axelerator. Experimental results show a large increase in performance while maintaining soundness of reachability results.

1 Introduction

Linear algebra packages have been developed in various flavours [1,17,23]. While the most formal of these packages use symbolic algorithms to ensure soundness, the most successful tools for large-scale applications sometimes sacrifice numerical soundness in favour of performance [13]. Similar trade-offs can be made in a number of related algorithms. For instance, eigenvalue problems frequently require matrices that are several orders of magnitude larger than those that symbolic evaluation can handle, and thus, are typically solved using floating-point calculations.

Floating-point computations cover a very wide range of problems, and are orders of magnitude faster than both symbolic and rational arithmetic. They are however, subject to rounding errors, which presents a number of challenges [16,27] that need to be addressed in order to obtain valid results.

The first problem we will focus on is that of soundness. Once an unknown error has been introduced, it is impossible to establish the correctness of the answer (although in most cases it will be correct to a given accuracy, we have no way to prove or disprove it). This problem can be solved with the use of interval arithmetic. By rounding outwards (i.e., increasing the size of the interval to include the error), we ensure that the true answer is always contained inside

* The authors were in part supported by ERC project 280053 (CPROVER), the H2020 FET OPEN 712689 SC^2, and the Alan Turing Institute, London, UK.

the interval, thus soundness is preserved. Using interval arithmetic can typically increase the computation time by a factor of four (see Section 6), which is moderate compared to the speed-up provided by the use of floating points.

The next problem we face regarding rounding errors is that they are cumulative. Each numerical operation performed will typically increase the error by a small amount, leading to a significant error after a large number of operations. When using intervals, this means that although the true result may be contained by the answer, the over-approximation could be large enough as to make this result meaningless. This means that some problems will require higher precision than others (in practice, a multiple precision arithmetic package will allow us to select an arbitrarily large precision). This comes at a cost dependent on the selected precision, so it is important to select an appropriate value for each problem. Algorithms requiring less iterations are therefore preferred to minimise the precision requirements.

The final challenge presented by the use of interval arithmetic appears when using comparisons to make binary decisions. When intervals partially intersect, it may not always be clear what the result of such comparisons should be. Therefore, we need to establish an order for the interval arithmetic.

In the following, we will discuss the above concepts with respect to a representative algorithm that benefits from the use of numerical algorithms. We use Abstract Acceleration [6, 21], which is a method that relies on solving linear programs in their corresponding eigenspaces. Since we look to study large dimensional problems, we work on a convex polyhedral domain, which in the case of 1-dimensional models reduces to standard interval arithmetic.

The main contribution of this work is to develop a sound numerical algorithm for Abstract Acceleration taking care of the errors introduced by the computations at each step. This goal entails the following:

1. We develop a numerical Simplex with error bounds using interval representations that ensures results are sound over-approximations of the original problem, i.e., the true results are always contained in the intervals. We note that, unlike previous work on sound linear solvers, our algorithm can reason about problems with pre-existing bounded errors in the problem statement.
2. We develop a Vertex Enumerator using intervals that ensures that all possible vertices of the original polyhedron (i.e., the expected polyhedron that would be obtained using exact arithmetic) are found within the hypercubes representing each abstracted vertex.
3. We develop a fast algorithm for describing eigenspaces of matrices within known error bounds. While this can largely be achieved using existing packages, the need to integrate multiple precision numbers with interval arithmetic and dealing with Jordan forms while maintaining a reasonable speed has motivated the development of a slightly different implementation.
4. We implement these techniques in the software tool Axelerator.[3]

[3] `www.cprover.org/LTI`

2 Preliminaries

Our work relies heavily on Interval Arithmetic and Convex Polyhedra, which are defined in the following.

Definition 1. *An interval domain is an abstract domain over the reals that defines a set of numbers by an upper and lower bound. We use the notation:*

$$[x] = [\underline{x}\ \overline{x}] = \{x \in \mathbb{R} \mid \underline{x} \leq x \leq \overline{x}\} \tag{1}$$

$$]x[=]\underline{x}\ \overline{x}[= \{x \mid \underline{x} < x < \overline{x}\} : x \in \mathbb{R} \tag{2}$$

$$[x[= [\underline{x}\ \overline{x}[= \{x \mid \underline{x} \leq x < \overline{x}\} : x \in \mathbb{R} \tag{3}$$

to represent an element of the domain. We also define the operators $+, -, *$[4]$, /$ *as the sum, subtraction, multiplication and division of an interval, with the following properties:*

$$[x_1] + [x_2] = [(\underline{x}_1 + \underline{x}_2) \quad (\overline{x}_1 + \overline{x}_2)], \tag{4}$$

$$[x_1] - [x_2] = [(\underline{x}_1 - \overline{x}_2) \quad (\overline{x}_1 - \underline{x}_2)], \tag{5}$$

$$[x_1] * [x_2] = [\inf(\underline{x}_1\underline{x}_2, \underline{x}_1\overline{x}_2, \overline{x}_1\underline{x}_2, \overline{x}_1\overline{x}_2) \quad \sup(\underline{x}_1\underline{x}_2, \underline{x}_1\overline{x}_2, \overline{x}_1\underline{x}_2, \overline{x}_1\overline{x}_2)], \tag{6}$$

$$\frac{[x_1]}{[x_2]} = \left[\inf\left(\frac{\underline{x}_1}{\underline{x}_2}, \frac{\underline{x}_1}{\overline{x}_2}, \frac{\overline{x}_1}{\underline{x}_2}, \frac{\overline{x}_1}{\overline{x}_2}\right) \quad \sup\left(\frac{\underline{x}_1}{\underline{x}_2}, \frac{\underline{x}_1}{\overline{x}_2}, \frac{\overline{x}_1}{\underline{x}_2}, \frac{\overline{x}_1}{\overline{x}_2}\right)\right] : 0 \notin [\underline{x}_2 \quad \overline{x}_2]. \tag{7}$$

Note that division is not defined for denominator intervals containing 0*. This case never applies to our algorithms since pivots, which is the only operation requiring division, operate on non-zero values. Additionally, any monotonic function over the reals can be mapped into an interval equivalent as:*

$$f(x) \to [f]([x]) = [\inf(f(\underline{x}), f(\overline{x})) \quad \sup(f(\underline{x}), f(\overline{x}))]. \tag{8}$$

In the case of non-monotonic functions, such as trigonometric operations, the calculation of $[f]([x])$ *has to take into consideration the minima and maxima of the function over the interval* $[x]$*.*

The above definitions are over the reals, which means that the result of any of these operations is exact. However, when using Finite Word Length representations such as floating points, results may not be representable in the given format and are implicitly rounded. In order for interval arithmetic to be sound, these domains must always round the upper limit upwards and the lower limit downwards, thus expanding the interval. These operations are well defined within the IEEE-754 standard [19] as well as in existing multiple precision libraries, such as mpfr [12], so we will not discuss the details here. For a full description of interval arithmetic over IEEE-754 floating point representations see [30].

Definition 2. *The support function of a point* $\boldsymbol{x} \in \mathbb{R}^n$ *in a given direction* $\boldsymbol{v} \in \mathbb{R}^n$ *is a scalar*

$$\rho_{\boldsymbol{x}}(\boldsymbol{v}) = \boldsymbol{x} \cdot \boldsymbol{v}, \tag{9}$$

[4] As standard, we will often omit this operator in the following.

where $\cdot$ is the dot product between vectors. This can be extended to set X, so that $\rho_X(\boldsymbol{v}) = \sup_{\boldsymbol{x} \in X} \boldsymbol{x} \cdot \boldsymbol{v}$.

Definition 3. *A convex polyhedron is an intersection of half-planes in $\mathbb{R}^n$ that is closed and non-empty. It can be described by the equation $\boldsymbol{C}\boldsymbol{x} \leq \boldsymbol{d} : \boldsymbol{C} \in \mathbb{R}^{m \times n}, \boldsymbol{x} \in \mathbb{R}^n, \boldsymbol{d} \in \mathbb{R}^m$. Each row in $\boldsymbol{C}$ corresponds to a vector normal to a bounding hyperplane of the polyhedron, while the corresponding element in $\boldsymbol{d}$ is the* support function *of the hyperplane. The hyperplane consists of all of the points $\boldsymbol{x}$ that meet the criterion $\boldsymbol{C}_i\boldsymbol{x} = \boldsymbol{d}_i$ (where i denotes the i-th component, and $\boldsymbol{C}_i$ the corresponding row).*

3 Related work

There are several tools and approaches using numeric computations for reachability analysis of large scale systems. Tools such as FLOW [8] or COSY [31] use Taylor models with remainder errors to calculate reach sets of non-linear systems. Floating point errors are over-approximated by pre-defined metrics on the calculations. SpaceEx [13] uses linear program analysis to rigorously evaluate reach sets. It also offers an option using support functions with numerically unsound calculations to achieve fast results. Similarly, reachability analysis tools relying on numeric algorithms contained in MATLAB [25] are likely to be unsound since error bounds are not typically provided in most of the enclosed algorithms. Set based simulations, like HYSON [3] or HYLAA [2] use set representations such as zonotopes and polyhedra to evaluate reach tubes.

Although a large number of algorithms have been developed to find rigorous bounds for linear equalities [24], there are not as many studies doing the same for optimal solutions over linear inequalities (which is the basis for Polyhedral Analysis). Polyhedral Analysis (as used in [6, 13, 20]) may require linear decision procedures to find solutions. These procedures would all benefit from numeric implementations with error bounds that meet the requirements for sound over-approximations.

There are historically two decision procedures that are commonly used for linear arithmetic: Fourier-Motzkin elimination and the simplex method. Most SMT solvers implement the latter because of its better performance when using rational arithmetic, while linear abstraction domains favor it for its performance on linear optimization problems. The original simplex has been revised numerous times; in particular, there is a variant that exploits duality for correction of accumulated errors in the case of unsound or imprecise arithmetic. Although some tools favor unsound implementations [13], there are numerous use-cases where the need for soundness prohibits the use of floating-point arithmetic.

Two solutions have arisen to address this problem. The first one [26] uses a fast floating-point based implementation of simplex to traverse the vertices of a polyhedron and then follows the selected pivots by using sound rational arithmetic along the same path. If the fast method produces a wrong result, the rational simplex continues its computation (thus losing the speed advantage) until it finds the correct solution.

The second method [7] uses interval arithmetic in Fourier-Motzkin elimination to over-approximate the solution for abstract polyhedra domains. Since abstract domains are typically over-approximations of the concrete domains they abstract, exact solutions are not necessary, and these over-approximations are acceptable. These two methods show an improvement in speed performance three- to tenfold, in most case studies. However, the analysis done in [29] shows that for a large number of real-world programs evaluated by openSMT, using an unsound floating-point simplex is marginally faster (13%) than the exact rational simplex. This is due to some optimizations in the handling of rationals inside the SMT solver and the fact that most benchmarked cases do not require higher precision arithmetic. However, we are interested in optimality as opposed to feasibility, which due to the larger number of pivots is likely trigger this case more frequently. For this reason, many of the methods and analyses presented for the SMT case are not necessarily well-suited for our use-case.

The algorithm in [15] provides an alternative: an iterative refinement method provides means to select the precision of the result arbitrarily. While this does not directly imply soundness, it can provide alternatives for $\delta-$checks to ensure it, though as all previously mentioned papers, it only addresses the problem of floating-point rounding errors. One of the problems presented here is that there are cases where imprecision appears even before calling the decision procedure. In the case of polyhedral abstractions, linear transformations may cause the numerical representations to grow in the rationals long before the objective function is called. We particularly refer to dynamics where primal components such as eigenvalues are extracted using numerical algorithms with bounded errors which are therefore present at their injection on the simplex. Similarly, there are cases where the actual program will have limitations in the representation of certain numbers which must be evaluated as small intervals. While a precise simplex is capable of performing such operations, the combinatorial explosion can render the problem intractable, and the use of an interval floating point simplex is evidently more efficient.

Error bounds for linear and mixed integer programming have been researched in [28]. The authors make use of interval arithmetic, and more specifically rounding modes, to find error bounds on optimal outcomes of the problem. Their approach allows the use of off-the-shelf algorithms for solving a linear problem using pre- and post- processing to deal with the errors. Unfortunately, this approach excludes the possibility of missing pivots, which means that the maximal vertex may not be reached. This can cause an unnecessarily large error bound to be discovered due to the symmetry in their calculation. Additionally, they do not address problems where the statement is already in interval form due to pre-transformations. To the best of the authors' knowledge, there are no existing techniques for sound numerical analysis in a polyhedral domain suitable for dealing with pre-existing errors in pipelined modular processes.

In addition to the linear programming operations, we note that many reachability tools depend on eigen-analysis to find solutions [6, 13, 21]. In fact, tools that verify continuous systems through discretization require such a component. Numerical analysis offers an incredible increase in speed performance. With respect to the eigen-decomposition itself, the most common approaches to estimating the error are those implemented in LAPACK [1]. These are based on the separation between eigenvalues and offer moderately tight bounds for both eigenvalues and eigenvectors. They do not however deal well with multiplicity since the formulas depend on condition numbers which become increasingly large when the gap between two eigenvalues is small. A formal approach to eigenvalues with algebraic multiplicities is offered by [32]. The authors provide a sound algorithm for finding the boundaries of both eigenvalues and eigenvectors based on an iterative estimation of the sub-eigenvector matrix relating to a cluster of eigenvalues. Their main limitation lies in the fact that the iterations are computationally as expensive as finding the estimations themselves, which is not a bad tradeoff. Since these values are often very small (typically insignificant with respect to other over-approximation errors), we choose speed over precision, and may compensate by using more precise data types at a lower time cost overall.

4 Abstract Acceleration

The problem statement we use to drive our discussion is that of Abstract Acceleration [6, 21]. Given an iterative program with dynamics

$$\boldsymbol{x}_{k+1} = \boldsymbol{A}\boldsymbol{x}_k : \boldsymbol{x}_k \in \mathbb{R}^n \wedge \boldsymbol{A} \in \mathbb{R}^{n\times n} \wedge k \in [0 \;\; \infty[, \tag{10}$$

we want to find all possible states visited over an infinite time horizon starting at an initial set X_0. We call this set the *reach tube* of the model:

$$\hat{X} = \{\boldsymbol{x}_k : k \in [0 \;\infty[\wedge \boldsymbol{x}_0 \in X_0 \wedge \boldsymbol{x}_{k+1} = \boldsymbol{A}\boldsymbol{x}_k\}. \tag{11}$$

It is easy to see that the infinite number of iterations would result in an unbounded cumulative rounding error, but using Abstract Acceleration we transform this problem into a one-step solution. We first change the problem statement by means of acceleration into

$$\hat{X} = \{\boldsymbol{x}_k = \boldsymbol{A}^k\boldsymbol{x}_0 : k \in [0 \;\infty[\wedge \boldsymbol{x}_0 \in X_0\}, \tag{12}$$

and further define new model semantics and corresponding set:

$$\hat{X} \subseteq \hat{X}^\sharp = \mathcal{A}X_0, \text{ such that } \bigcup_{k\in[0 \;\; \infty[} \boldsymbol{A}^k \subseteq \mathcal{A}, \tag{13}$$

where the newly introduced *abstract reach tube* $\hat{X}^\sharp$ is an over-approximation of the reach tube of the model [21]. All we need to do is define the nature of $\mathcal{A}X_0$.

Let $\boldsymbol{S}\boldsymbol{J}\boldsymbol{S}^{-1} = \boldsymbol{A}$ be the eigen-decomposition of $\boldsymbol{A}$, where $\boldsymbol{S}$ is the set of generalised eigenvectors of $\boldsymbol{A}$ and $\boldsymbol{J}$ is the Jordan form containing the eigenvalues of $\boldsymbol{A}$ in its main diagonal. We correspondingly define $\mathcal{J} = \boldsymbol{S}^{-1}\mathcal{A}\boldsymbol{S}$, which is such that $\mathcal{J} \supseteq \bigcup_{k\in[0 \;\; \infty[} \boldsymbol{J}^k$.

Given a set $X = \{\boldsymbol{x} : \boldsymbol{C}\boldsymbol{x} \leq \boldsymbol{d}\}$. The linear transformation $\boldsymbol{x}' = \boldsymbol{S}^{-1}\boldsymbol{x}$ leads to set

$$X' = \boldsymbol{S}^{-1}X = \{\boldsymbol{x}' : \boldsymbol{C}\boldsymbol{S}\boldsymbol{x}' \leq \boldsymbol{d}\}. \tag{14}$$

Analogously, let $X'_0 = \boldsymbol{S}^{-1}X_0$ and $\hat{X}'^{\sharp} = \boldsymbol{S}^{-1}\hat{X}^{\sharp}$ be the initial set and the abstract reach tube mapped onto the eigenspace of matrix $\boldsymbol{A}$. We may transform equation (13) into:

$$\hat{X}' \subseteq \hat{X}'^{\sharp} = \mathcal{J}X'_0, \text{ such that } \bigcup_{k\in[0\ \ \infty[} \boldsymbol{J}^k \subseteq \mathcal{J}, \tag{15}$$

where $\hat{X}' = \boldsymbol{S}^{-1}\hat{X}$ and $\mathcal{J}$ is a convex polyhedron representing restrictions on the eigenvalues of $\boldsymbol{A}$. $\hat{X}'^{\sharp}$ is calculated by applying a simplex algorithm with a tableau defined by $\mathcal{J}$ and objective functions derived from each vertex of X'_0 (step (10) in Algorithm 1 below). We note, however, that if we wish to use numerical analysis in the decomposition of $\boldsymbol{A}$, the Tableau in $\mathcal{J}$ contains small intervals given by the errors, which cannot be processed efficiently by a regular simplex and require a new procedure. Also note that the error bounds for $\boldsymbol{S}$ will have a similar effect on the objective functions.

The calculation of the abstract reach tube via abstract acceleration is encompassed in Algorithm 1, which can be summarised as follows (its steps are denoted in parentheses):

1. We perform unsound eigen-decomposition using an existing algebra package (lines 2-3).
2. Then we restore soundness to the results using the methods described in Section 5.2 (lines 4–5).
3. The inverse of the matrix of eigenvectors is calculated after soundness is restored, using interval arithmetic in order to ensure its soundness (line 6).
4. The abstract dynamics are obtained by evaluating the convex hull of all powers of eigenvalues up to the desired number of iterations as described in [6] (line 7).
5. Using Equation (14), transform the initial state into the eigenspace (line 8).
6. Extract the vertices of the eigen-polyhedron X'_0 using the sound over-approximation of the double description algorithm [14] from Section 5.4 (line 9). It is worth noting that this algorithm uses a simplex to seed it: we will first discuss the simplex algorithm, which is also needed at the next step, and then the vertex enumeration algorithm used at this stage.
7. Calculate the mapped *abstract reach tube* $\hat{X}'^{\sharp}$ that over-approximates the image of the reach tube in the eigenspace (line 10). This is achieved by evaluating a set of objective functions using the abstract dynamics as a simplex Tableau, and via a sound simplex described in Section 5.3. The objective functions are defined by the vertices of X'_0, denoted as V_0, and by the desired template directions, such that

   $$\boldsymbol{w}_{ij} = \boldsymbol{v}_i \circ \boldsymbol{t}_j : \boldsymbol{v}_i \in V_0 \wedge \boldsymbol{t}_j \in T,$$

   where T is the set of template directions, and $\circ$ denotes a component-wise multiplication yielding a vector.

8. Using Equation (14), find the reach tube $\hat{X}^\sharp = \boldsymbol{S}\hat{X}'^\sharp$ (line 11).

Algorithm 1 Calculation of Abstract Reach Tube Using Abstract Acceleration

Input: $X_0\boldsymbol{A}, k$.
Output: $\hat{X}^\sharp$

```
1:  function findAbstractReachTube()
2:      $\hat{\boldsymbol{J}}$ = calculateEigenvalues($\boldsymbol{A}$)
3:      $\hat{\boldsymbol{S}}$ = calculateEigenvectors($\boldsymbol{A}$)
4:      $[\boldsymbol{J}]$ = soundifyEigenvalues($\boldsymbol{A}, \hat{\boldsymbol{J}}, \hat{\boldsymbol{S}}$)
5:      $[\boldsymbol{S}]$ = soundifyEigenvectors($\boldsymbol{A}, [\boldsymbol{J}], \hat{\boldsymbol{S}}$)
6:      $[\boldsymbol{S}]^{-1}$ = calculateInverse($[\boldsymbol{S}]$)
7:      $\mathcal{J}$ = getAbstractDynamics($[\boldsymbol{J}]$)
8:      $[X'_0]$ = transformInitialSpace($X_0, [\boldsymbol{S}]^{-1}$)
9:      $[V_0]$ = getVertices($[X'_0]$)
10:     $\hat{X}'^\sharp$ = getReachTube($\mathcal{J}, [V_0]$)
11:     $\hat{X}^\sharp$ = transformReachTube($\hat{X}'^\sharp, [\boldsymbol{S}]$)
12: end function
```

5 Implementation

5.1 An Interval Partial Order for Vector Spaces

Interval arithmetic is widely researched and used in the literature. The main issue in its implementation here is with respect to ordering, since some of the operations used require ordered sets. While real numbers have a well defined order, there are several different options regarding the order of real intervals. This creates a problem for programs dealing with branching since an incorrect assumption on the order will cause undetermined behaviour. The reader is referred to literature [5, 9–11] for possible orderings of real intervals. In this paper, we have selected the following paradigm:
Let $[x] = [\underline{x}\ \overline{x}]$ be an interval of real numbers such that

$$\begin{cases} [x] < 0 & \overline{x} < 0 \\ [x] > 0 & \underline{x} > 0 \\ [x] = 0 & -e \leq \underline{x} \leq 0 \wedge 0 \leq \overline{x} \leq e \\ [x] \text{ is deemed imprecise} & \underline{x} < -e \wedge \overline{x} \geq 0 \vee \underline{x} \leq 0 \wedge \overline{x} > e, \end{cases} \quad (16)$$

where e is a user-defined error bound.

The first two definitions correspond to precedence definitions in the IEEE standard, but the third one is an enabling comparison (*i.e.* it corresponds to "may be equal to") which is not present in the standard. Throughout this work this latter definition is useful because the operations that relate to this condition have no negative effect if applied when the condition is false (apart from the increased processing time of the operation), nor do they compromise soundness.

Imprecise numbers break the ordering: for example, they could be originally non-zero elements whose error touches zero – while this situation does not break the soundness of the algorithms described in this paper, it affects their precision and completeness, as described in Section 5.3). As such, it is established that the appearance of an imprecise number forces a change in the error bound e or an increase in the precision, so that the accumulated errors do not surpass this bound (e). From the above paradigm we can easily derive that

$$\begin{cases} [x] = [y] \iff [x] - [y] = 0 \\ [x] < [y] \iff [x] - [y] < 0 \\ [x] > [y] \iff [x] - [y] > 0, \end{cases} \tag{17}$$

which results in our ordering.

We will extend this definition to the dot product of two vectors, in order to establish equality in higher-dimensional spaces (in particular, this will allow us to set an ordering of value pairs). Let $\mathbb{v} = [[v_1] \cdots [v_n]]^T$ and $\mathbb{u} = [[u_1] \cdots [u_n]]^T$ be interval column vectors, with $[d] = \mathbb{v} \cdot \mathbb{u}$, then we say that

$$\begin{cases} \mathbb{v} \cdot \mathbb{u} < 0 & \overline{d} < 0 \\ \mathbb{v} \cdot \mathbb{u} > 0 & \underline{d} > 0 \\ \mathbb{v} \cdot \mathbb{u} = 0 & -e \leq \underline{d} \leq 0 \wedge 0 \leq \overline{d} \leq e \\ \mathbb{v} \cdot \mathbb{u} \text{ is deemed imprecise} & \text{otherwise.} \end{cases} \tag{18}$$

5.2 Eigen-Decomposition

The first stage required for Abstract Acceleration is the eigen-decomposition of the dynamics (steps 2-3 in Algorithm 1). We seek to find the Jordan form of a matrix, characterising its eigenvalues alongside their corresponding eigenvectors, with known error bounds for both. For this purpose we use the package *eigen* [17], which contains an efficient fast numerical eigensolver using Schur decomposition. The main advantage of eigen over other packages is that it is a template library which can be used with any numerical data type. Since we are interested in using multiple precision floating point integers, this is an important feature of the desired package. Unfortunately, the eigen-decomposition cannot be performed using interval arithmetic. This is because numerical solvers for this problem exploit the convergence of successive results towards a precise value. While the process is known to converge, the latter iterations will typically oscillate around the final values, which causes the interval containing the final eigenvalue to expand, resulting in a width that becomes unbounded, rather than in a small interval around the expected result. We therefore use standard arithmetics to obtain a numerical approximation of the true eigenspace, and then find error bounds using interval arithmetic to generate the intervals containing the true values in the eigenspace. Namely, we call a standard unsound eigen-decomposition algorithm, and later make it sound by creating intervals around the results using soundly-calculated error bounds.

We remark that these error bounds are found in the LAPACK [1] package, used by programs such as Matlab, and could therefore be correctly obtained by using this tool. However, four issues drive us away from LAPACK. The first is that the library is written in FORTRAN and uses double-precision floating-point arithmetic: this restricts our ability to use higher precision to obtain smaller errors. The second is that some of the procedures used in LAPACK to calculate the error can be time consuming, which can have a large impact on the overall processing time. The third one is that LAPACK does not allow the use of intervals, hence the operations that calculate the error bound have rounding errors themselves and are thus unsound. The final problem is that LAPACK does not provide Jordan forms with geometric multiplicities, nor can it always ensure the error bounds for algebraic multiplicities greater than one.

The calculation of the error bounds for soundness is performed in two stages, first for the eigenvalues and then for the eigenvectors (the latter requires the sound eigenvalue intervals in its calculation).

We first define an interval matrix, which will be used during both stages. Let

$$\mathbb{M} \in \mathbb{R}^{p \times q} = \begin{bmatrix} [m_{11}] & \cdots & [m_{1q}] \\ \vdots & \ddots & \vdots \\ [m_{p1}] & \cdots & [m_{pq}] \end{bmatrix}, \tag{19}$$

where $[m_{ij}] = \left[\underline{m_{ij}} \quad \overline{m_{ij}}\right] : i \in [1 \dots p] \wedge j \in [1 \dots q]$ be an interval matrix. Interval arithmetics between interval matrices derives directly from the operations between their elements. We may trivially construct an interval equivalent of any non-interval matrix using the following definition:

$$\mathbb{M} = \boldsymbol{M} \text{ iff } \forall [m_{ij}] \in \mathbb{M}, \;\; \underline{m_{ij}} = \overline{m_{ij}} = m_{ij} \in \boldsymbol{M} : i \in [1 \dots p] \wedge j \in [1 \dots q]. \tag{20}$$

Error Bounds on the Eigenvalues (step (4) in Algorithm 1).

Theorem 1. *Given an eigenvalue λ_i with algebraic multiplicity m_i obtained using Jordan decomposition, the error of the numerically calculated eigenvalue $\hat{\lambda}_i$ is upper bounded by the formula:*

$$e_{\lambda_i} \le e_{m_i} = \begin{cases} E & m_i = 1 \\ \frac{E^{\frac{1}{m_i}}}{1 - E^{\frac{1}{m_i}}} & m_i > 1 \end{cases}, \text{ where } E = \max\left(k\left(\hat{\mathbb{S}}\right) \|\hat{\mathbb{S}}\hat{\mathbb{J}}\hat{\mathbb{S}}^{-1} - \mathbb{A}\|_2\right), \tag{21}$$

where m_i is the geometric multiplicity of λ_i, $\hat{\mathbb{S}} = \hat{\boldsymbol{S}}$ are the calculated eigenvectors of $\boldsymbol{A}$, $\hat{\mathbb{J}} = \hat{\boldsymbol{J}}$ its calculated Jordan form and $k(\mathbb{M})$ is the condition number of a given matrix $\mathbb{M}$ that is defined as $[\sigma_{max}](\mathbb{M})/[\sigma_{min}](\mathbb{M})$, where $[\sigma_i]$ are the singular values of $\mathbb{M}$.[5] *The obtained matrix $\mathbb{J} = [\hat{\boldsymbol{J}} - \sup(e_{m_i})\boldsymbol{I} \quad \hat{\boldsymbol{J}} + \sup(e_{m_i})\boldsymbol{I}] \supseteq \boldsymbol{J} : i \in [1 \dots n]$ is a sound over-approximation of the diagonal matrix with the eigenvalues of $\boldsymbol{A}$. All matrices are in $\mathbb{R}^{n \times n}$.*

[5] Note that this is equivalent to $k(\mathbb{M}) = \|\mathbb{M}\|_2 \|\mathbb{M}^{-1}\|_2$.

Proof. Let us first assume that matrix $\boldsymbol{A}$ is diagonalizable (an hypothesis relaxed below). Let us also define the numerically calculated approximation $\hat{\boldsymbol{A}} \simeq \boldsymbol{A}$ with Jordan form $\hat{\boldsymbol{J}}$ and eigenvectors $\hat{\boldsymbol{S}}$. Then the error bound for each of the eigenvalues is [33]:

$$e_{\lambda_i} = |\lambda_i - \hat{\lambda}_i| < k(\boldsymbol{S})\|\boldsymbol{A} - \hat{\boldsymbol{A}}\|_2 = k(\boldsymbol{S})\|\boldsymbol{A} - \hat{\boldsymbol{S}}\hat{\boldsymbol{J}}\hat{\boldsymbol{S}}^{-1}\|_2. \tag{22}$$

Note that symmetrically

$$e_{\lambda_i} = e_{\hat{\lambda}_i} = |\hat{\lambda}_i - \lambda_i| < k(\hat{\boldsymbol{S}})\|\hat{\boldsymbol{A}} - \boldsymbol{A}\|_2 = k(\hat{\boldsymbol{S}})\|\hat{\boldsymbol{S}}\hat{\boldsymbol{J}}\hat{\boldsymbol{S}}^{-1} - \boldsymbol{A}\|_2 \tag{23}$$

as long as $\hat{\boldsymbol{A}} = \hat{\boldsymbol{S}}\hat{\boldsymbol{J}}\hat{\boldsymbol{S}}^{-1}$ has no rounding errors. Therefore, to ensure soundness, we must translate the error calculation into an interval arithmetic problem.

Using (20), Equation (23) then becomes

$$e_{\lambda_i} < \sup\left(k\left(\hat{\mathbb{S}}\right)\|\hat{\mathbb{S}}\hat{\mathbb{J}}\hat{\mathbb{S}}^{-1} - \mathbb{A}\|_2\right).$$

For simplicity, we will hereon use $\hat{\mathbb{A}} = \hat{\mathbb{S}}\hat{\mathbb{J}}\hat{\mathbb{S}}^{-1}$.

We could calculate tighter bounds for each eigenvalue using the condition number of individual eigenvectors (which is defined for non-square matrices as $k(\boldsymbol{M}) = \|\boldsymbol{M}\|_2\|\boldsymbol{M}^+\|_2$, where $\boldsymbol{M}^+$ is the pseudo-inverse of $\boldsymbol{M}$), but we choose this faster approach expecting the increased error to be negligible with respect to the dynamics.

Extending our analysis by relaxing the assumption made above, when there exists a Jordan block in $\boldsymbol{A}$ with geometric multiplicity m_i, then the error can be derived by leveraging [33] as follows:

$$\begin{aligned}
&\tfrac{|\lambda_i-\hat{\lambda}_i|^{m_i}}{(1+|\lambda_i-\hat{\lambda}_i|)^{m_i-1}} = (1+|\lambda_i-\hat{\lambda}_i|)\left(\tfrac{|\lambda_i-\hat{\lambda}_i|}{(1+|\lambda_i-\hat{\lambda}_i|)}\right)^{m_i} < k(\hat{\boldsymbol{S}})\|\hat{\boldsymbol{S}}\hat{\boldsymbol{J}}\hat{\boldsymbol{S}}^{-1} - \boldsymbol{A}\|_2 \\
&\Rightarrow \tfrac{|\lambda_i-\hat{\lambda}_i|}{(1+|\lambda_i-\hat{\lambda}_i|)} < \left(k(\hat{\boldsymbol{S}})\|\hat{\boldsymbol{S}}\hat{\boldsymbol{J}}\hat{\boldsymbol{S}}^{-1} - \boldsymbol{A}\|_2\right)^{\frac{1}{m_i}} < \sup\left(k(\hat{\mathbb{S}})\|\hat{\mathbb{A}} - \mathbb{A}\|_2\right)^{\frac{1}{m_i}} \\
&\Rightarrow e_{\lambda_i} < \sup\left(\frac{\left(k(\hat{\mathbb{S}})\|\hat{\mathbb{A}}-\mathbb{A}\|_2\right)^{\frac{1}{m_i}}}{1-\left(k(\hat{\mathbb{S}})\|\hat{\mathbb{A}}-\mathbb{A}\|_2\right)^{\frac{1}{m_i}}}\right).
\end{aligned}$$

However, this bound requires that the correct Jordan shape be selected (i.e., the one that corresponds to the original dynamics without numerical errors), which means we need to use the formula using the largest possible Jordan block for each set of similar λ (i.e., eigenvalues which intersect given their error intervals). In fact, this is not enough to ensure the bound since different shapes will result in different condition numbers (since they will have different generalized eigenvectors), so we are forced to calculate the maximum bound for all options. We will see later how to overcome this difficulty in a more efficient way. □

Now that we can obtain sound eigenvalues, we will proceed to restore soundness to the eigenvectors.

Error Bounds on the Eigenvectors (step 5 in Algorithm 1).

Theorem 2. *The interval eigenvector*

$$\mathbb{v}_i = \left[\hat{\mathbb{v}}_i \cos([\theta]_i) \quad \frac{\hat{\mathbb{v}}_i}{\cos([\theta]_i)}\right] \supseteq \boldsymbol{v}_i, \text{ where} \tag{24}$$

$$[\theta]_i \leq [\hat{\theta}]_i = \left(\frac{n}{n-1}\right)^{\frac{n-1}{2}} \frac{\|\mathbb{A} - \hat{\mathbb{A}}]\|_2 \left(n\|\mathbb{U} - [\lambda_i]\boldsymbol{I}\|_2\right)^{n-1}}{n^{\frac{n}{2}} \prod_{i\neq j} \left([\lambda_j] - [\lambda_i]\right)^{m_i}} : [\hat{\theta}]_i < \frac{\pi}{2}$$

is an over-approximation of the true eigenvector $\boldsymbol{v}_i$. Here, n is the dimension of $\boldsymbol{A}$, m_i is the size of the i^{th} Jordan block of $\boldsymbol{A}$, $\hat{\mathbb{v}}_i = \hat{\boldsymbol{v}}_i$ the numerically calculated i^{th} eigenvector of $\boldsymbol{A}$, $[\lambda_i]$ the error-bound interval for the i^{th} eigenvalue of $\boldsymbol{A}$ (inherited from above), and $\mathbb{U} = \mathbb{Q}^{-1}\mathbb{A}\mathbb{Q}$ where $\mathbb{Q} = \boldsymbol{Q}$ the Schur decomposition of $\boldsymbol{A}$.

Given sufficient precision in the numerical calculations, we have that $[\hat{\theta}]_i < \frac{\pi}{2}$. This inequality can always be obtained by increasing precision.

Proof. The error angle between the numerically calculated i^{th} eigenvector and the true i^{th} eigenvector of $\boldsymbol{A}$ is

$$\theta_i = \cos^{-1}(\boldsymbol{v}_i \cdot \hat{\boldsymbol{v}}_i) < \frac{\|\boldsymbol{A} - \hat{\boldsymbol{S}}\hat{\boldsymbol{J}}\hat{\boldsymbol{S}}^{-1}\|_2}{sep_i} : \|\boldsymbol{v_i}\|_2 = \|\hat{\boldsymbol{v}}_i\|_2 = 1, \tag{25}$$

where $\boldsymbol{v}_i$ is the original i^{th} eigenvector, $\hat{\boldsymbol{v}}_i$ is the numerically calculated eigenvector, and sep_i is the separation between Jordan blocks, which is calculated as follows.

Let $\boldsymbol{U}$ be an upper triangular matrix such that $\boldsymbol{AQ} = \boldsymbol{QU}$ with $\boldsymbol{Q}$ a unitary matrix ($\boldsymbol{Q}^{-1} = \boldsymbol{Q}^*$). This is the Schur decomposition of $\boldsymbol{A}$. The eigenvalues of $\boldsymbol{A}$ are the diagonal entries of $\boldsymbol{U}$. There are $s!$ different matrices $\boldsymbol{U}$ (where s is the number of Jordan blocks) corresponding to all possible permutations of the eigenvalues of $\boldsymbol{A}$ in the diagonal. Let

$$\boldsymbol{U} = \begin{bmatrix} \boldsymbol{U}_{11} & \boldsymbol{U}_{12} \\ \boldsymbol{0} & \boldsymbol{U}_{22} \end{bmatrix}, \boldsymbol{U}_{11} \in \mathbb{R}^{m\times m}, \boldsymbol{U}_{22} \in \mathbb{R}^{(n-m)\times(n-m)},$$

such that the eigenvalues of $\boldsymbol{U}_{11}$ are the eigenvalues of the i^{th} Jordan block of $\boldsymbol{A}, \boldsymbol{J}_i \in \mathbb{R}^{m\times m}$. The separation sep_i of $\boldsymbol{J}_i$ is the smallest difference between any singular value of $\boldsymbol{U}_{11}$ and those of $\boldsymbol{U}_{22}$ [36]. This value can be obtained by computing the smallest singular value of the Kronecker product [22]

$$\boldsymbol{K} = \boldsymbol{U}_{11} \otimes \boldsymbol{I}_{(n-m),(n-m)} - \boldsymbol{I}_{m,m} \otimes \boldsymbol{U}_{22}.$$

However, this computation is expensive. Moreover, a permutation of the matrix $\boldsymbol{U}$ must be executed for each different eigenvalue of $\boldsymbol{U}$. Hence, we look for a solution that avoids computing these values altogether.

First we will find a lower bound for the separation, which can be obtained by applying [18] to the Kronecker product $\boldsymbol{K}$:

$$\sigma_{min}(\boldsymbol{K}) \geq \left(\frac{p-1}{p}\right)^{\frac{p-1}{2}} \det(\boldsymbol{K}) \min\left(\frac{c_{min}}{\prod_1^p c_j}, \frac{r_{min}}{\prod_1^p r_j}\right) : \boldsymbol{K} \in \mathbb{R}^{p\times p}, p = m(n-m), \tag{26}$$

where n is the dimension of the original matrix, m is the dimension of the current Jordan block, c_j is the 2-norm of the j^{th} column of $\boldsymbol{K}$ and r_j the 2-norm of the j^{th} row (with corresponding minima c_{min}, r_{min}).

Let us first look at the case of matrices with all eigenvalues having algebraic multiplicity of 1 (we'll use the apex i to indicate a partition of $\boldsymbol{U}$ relating to the i^{th} Jordan block). In this case $\boldsymbol{U}_{11}^i = \lambda_i$ and, since $\boldsymbol{K}^i$ is an upper triangular matrix and its determinant is therefore the product of its diagonal entries,

$$det(\boldsymbol{K}^i) = \prod_{i \neq j} (\lambda_j - \lambda_i).$$

We also note that

$$\sum_1^p c_j^2 = \|\boldsymbol{K^i}\|_2^2 \ \wedge \ \sum_1^p r_j^2 = \|\boldsymbol{K^i}\|_2^2.$$

Using the arithmetic and geometric mean inequality [35] we have

$$\prod_{j=1}^p c_j \leq \left(\frac{1}{p}\right)^{\frac{p}{2}} \|\boldsymbol{K^i}\|_2^p \ \wedge \ \prod_{j=1}^p r_j \leq \left(\frac{1}{p}\right)^{\frac{p}{2}} \|\boldsymbol{K^i}\|_2^p,$$

where $\|\boldsymbol{K^i}\|_2 = \|\boldsymbol{U_{22}^i} - \lambda_i \boldsymbol{I}\|_2 \leq \|\boldsymbol{U} - \lambda_i \boldsymbol{I}\|_2$.

Finally, given that for any matrix $\boldsymbol{U}_{11}^i$ we can select any permutation of $\boldsymbol{U}_{22}^i$ such that the first element of $\boldsymbol{K^i}$ is $\min_{i \neq j}(\lambda_j - \lambda_i)$ and given that $\boldsymbol{K^i}$ is upper triangular, this means that $c_{min} = \min_{i \neq j}(\lambda_j - \lambda_i) \leq r_{min}$.

Going back to Equation (26), we have:

$$\sigma_{min}(\boldsymbol{K^i}) \geq \left(\frac{p-1}{p}\right)^{\frac{p-1}{2}} \frac{\prod_{i \neq j}(\lambda_j - \lambda_i)}{\left(\frac{1}{p}\right)^{\frac{p}{2}} \|\boldsymbol{U} - \lambda_i \boldsymbol{I}\|_2^{p-1}}.$$

This term neither depends on the calculation of $\boldsymbol{K}$, nor on the ordering of $\boldsymbol{U}$.

In the case of a matrix $\boldsymbol{U}$ with algebraic multiplicity strictly greater than one, we should remark that the matrix $\boldsymbol{K}$ has dimension $m(n-m)$. Its determinant is

$$\det(\boldsymbol{K^i}) = \left(\prod_{i \neq j}(\lambda_j - \lambda_i)\right)^m$$

and its norm is

$$\|\boldsymbol{K^i}\|_2 \leq m\|\boldsymbol{U}_{22}^i - \lambda_i \boldsymbol{I}\|_2 + (n-m)\|\boldsymbol{U}_{11}^i - \lambda_i \boldsymbol{I}\|_2 \leq n\|\boldsymbol{U} - \lambda_i \boldsymbol{I}\|_2,$$

therefore

$$\sigma_{min}(\boldsymbol{K^i}) \geq \left(\frac{n-1}{n}\right)^{\frac{n-1}{2}} \frac{\left(\prod_{i \neq j}(\lambda_j - \lambda_i)\right)^m}{\left(\frac{1}{n}\right)^{\frac{n}{2}} \left(n\|\boldsymbol{U} - \lambda_i \boldsymbol{I}\|_2\right)^{n-1}}.$$

Replacing for (25) and using interval arithmetic we get Equation (24). The last part of the equation comes from the need for the cosine to be positive. In practice we want a much smaller number, so if θ_i is too large, then we can report that the result is imprecise and require the use of higher precision. $\square$

Error Bounds for unknown Jordan Shapes As stated earlier, the preceding discussion relies on having selected the correct Jordan shape in the first place (that is, the Jordan shape for the theoretical decomposition without calculation errors), which is in most cases unverifiable. This means that our solution thus far can only be fully sound for diagonalisable matrices (i.e., if the separation of the eigenvalues is larger than the error) or those where the Jordan shape is known a-priori. We therefore propose an additional mechanism to deal with the case of non-diagonalisable matrices with unknown Jordan shapes. In the following, the symbol $\mathbf{1} = \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix}$ represents a appropriately-sized matrix with elements all equal to one.

Theorem 3. *Given a numerical decomposition of* $\boldsymbol{A}$, $\hat{\boldsymbol{A}} = \hat{\boldsymbol{S}}\hat{\boldsymbol{J}}\hat{\boldsymbol{S}}^{-1}$, *the interval matrix*

$$\mathbb{A}'_k = \hat{\mathbb{S}}'\hat{\mathbb{J}}'_k\hat{\mathbb{S}}'^{-1} : \hat{\mathbb{J}}'_k = \hat{\boldsymbol{J}}^k + ((\|\hat{\boldsymbol{J}}\|_1 + [e])^k - \|\hat{\boldsymbol{J}}\|_1^k) \text{ and } \hat{\mathbb{S}}' = \hat{\boldsymbol{S}}(\boldsymbol{I} + [e]\mathbf{1}) \quad (27)$$

where $[e] = [-e \;\; e]$ *and* $e = \max(n, \|\hat{\boldsymbol{J}}\|)\|\hat{\boldsymbol{S}}^{-1}\boldsymbol{A}\hat{\boldsymbol{S}} - \hat{\boldsymbol{J}}\|_2$, *is an overapproximation of* $\boldsymbol{A}^k$.

Proof. Let

$$\hat{\boldsymbol{S}}^{-1}\boldsymbol{A}\hat{\boldsymbol{S}} = \hat{\boldsymbol{J}} + \boldsymbol{J}_e \Leftrightarrow \boldsymbol{A} = \hat{\boldsymbol{S}}(\hat{\boldsymbol{J}} + \boldsymbol{J}_e)\hat{\boldsymbol{S}}^{-1},$$

where $\boldsymbol{J}_e = \hat{\boldsymbol{S}}^{-1}\boldsymbol{A}\hat{\boldsymbol{S}} - \hat{\boldsymbol{J}}$ is an error matrix computed from the known quantities on the RHS of the equation. Then

$$\boldsymbol{A}^k = \hat{\boldsymbol{S}}(\hat{\boldsymbol{J}} + \boldsymbol{J}_e)^k\hat{\boldsymbol{S}}^{-1}. \quad (28)$$

Let $[e'] = [-\|\boldsymbol{J}_e\|_1 \;\; \|\boldsymbol{J}_e\|_1]$ and $\mathbb{J}'_e = [e']\mathbf{1}$, then $\boldsymbol{J}_e \subseteq \mathbb{J}'_e$.

Since each element of $\mathbb{J}'_e$ is $[e']$, each element in $\mathbb{J}'^k_e$ will be $n^{k-1}[e']^k = (n\|\boldsymbol{J}_e\|_1)^{k-1}[e']$, therefore $\mathbb{J}'^k_e = (n\|\boldsymbol{J}_e\|_1)^{k-1}[\boldsymbol{J}'_e]$. Similarly, $\mathbb{J}'_e\hat{\boldsymbol{J}}^k\mathbb{J}'_e \subseteq \|\boldsymbol{J}_e\|_1 \|\hat{\boldsymbol{J}}\|_1^k\mathbb{J}'_e$.

Let $[e] = \max(n, \|\hat{\boldsymbol{J}}\|_1)[e'] \wedge \mathbb{J}_e = [e]\mathbf{1}$, so that $(n\|\boldsymbol{J}_e\|_1)^{k-1}\mathbb{J}'_e \subseteq ([e])^{k-1}[\boldsymbol{J}'_e] \wedge \|\boldsymbol{J}_e\|_1\|\hat{\boldsymbol{J}}\|_1\mathbb{J}'_e \subseteq [e]\mathbb{J}'_e$. More generally any matrix multiplication with i elements $\hat{\boldsymbol{J}}$ and j elements $\mathbb{J}'_e$ may be overapproximated by $[e]^{j-1}\|\hat{\boldsymbol{J}}\|_1^i\mathbb{J}'_e \subseteq [e]^j\|\hat{\boldsymbol{J}}\|_1^i\mathbf{1}$. From the above properties we expand (28) replacing for these values and obtain:

$$\boldsymbol{A}^k \subseteq \hat{\boldsymbol{S}}\left(\hat{\boldsymbol{J}}^k + \sum_{i=0}^{k-1}\binom{k}{i}[e]^{k-i}\|\hat{\boldsymbol{J}}\|_1^i\mathbf{1}\right)\hat{\boldsymbol{S}}^{-1} \quad (29)$$

$$\Rightarrow \boldsymbol{A}^k \subseteq \hat{\boldsymbol{S}}\left(\hat{\boldsymbol{J}}^k - \|\hat{\boldsymbol{J}}\|_1^k + \sum_{i=0}^{k}\binom{k}{i}[e]^{k-i}\|\hat{\boldsymbol{J}}\|_1^i\mathbf{1}\right)\hat{\boldsymbol{S}}^{-1}$$

$$\Rightarrow \boldsymbol{A}^k \subseteq \hat{\boldsymbol{S}}\left(\hat{\boldsymbol{J}}^k + ((\|\hat{\boldsymbol{J}}\|_1 + [e])^k - \|\hat{\boldsymbol{J}}\|_1^k)\mathbf{1}\right)\hat{\boldsymbol{S}}^{-1}$$

$$\Rightarrow \boldsymbol{A}^k \subseteq \hat{\boldsymbol{S}}(\boldsymbol{I} + \mathbb{J}_e)\left(\hat{\boldsymbol{J}}^k + ((\|\hat{\boldsymbol{J}}\|_1 + [e])^k - \|\hat{\boldsymbol{J}}\|_1^k)\right)\hat{\boldsymbol{S}}^{-1}.$$

□

Notice that, since the formula depends on the horizon k, we cannot in general prove soundness for an unbounded time horizon in this case. In the instance of converging models (as is often the case for industrial systems), we may pick a k that practically reaches a fix point in finite time, thus extending the proof for an infinite time horizon.

5.3 Interval Simplex

The key implementation required for the Algorithm (step 10 in Algorithm 1) is a variant of simplex that can handle interval representations. We first remark that throughout this paper we are looking at over-approximations of desired quantities in order to ensure soundness, and to optimise algorithmic performance. Let us begin by exploring the meaning of a polyhedral description using interval inequalities. An interval polyhedron $[P] = \{\boldsymbol{x} : \mathbb{A}\boldsymbol{x} \leq \mathbb{b}\}$ is a union of polyhedra such that

$$[P] = \bigcup P_i : P_i = \{\boldsymbol{x} : \boldsymbol{A}_i\boldsymbol{x} \leq \boldsymbol{b}_i\}, \boldsymbol{A}_i \in \mathbb{A} \wedge \boldsymbol{b}_i \in \mathbb{b}. \tag{30}$$

Note that $[P]$ is not guaranteed to be convex even if all P_i are. We begin by simplifying this description.

Theorem 4. *The polyhedron*

$$\hat{P}_i = \left\{\boldsymbol{x} : \boldsymbol{A}_i\boldsymbol{x} \leq \hat{\boldsymbol{b}} \wedge \boldsymbol{A}_i \in \mathbb{A} \wedge \hat{\boldsymbol{b}}^j = \overline{\boldsymbol{b}^j}\right\}, \tag{31}$$

where $\mathbb{b}^j$ *is the* j^{th} *row of* $\mathbb{b}$ *and* $\overline{\boldsymbol{b}^j} = \sup(\mathbb{b}^j)$, *is a sound over-approximation of* P_i.

Proof. Let $\boldsymbol{r}_i^j$ be a row in $\boldsymbol{A}_i$ with $\rho_{P_i}(\boldsymbol{r}_i^j) = \boldsymbol{b}_i^j$ its corresponding support function. Equations (30) and (31) state that

$$\boldsymbol{x} \in P_i \Leftrightarrow \forall j,\ \boldsymbol{r}_i^j\boldsymbol{x} \leq \boldsymbol{b}_i^j \quad \wedge \quad \boldsymbol{x} \in \hat{P}_i \Leftrightarrow \forall j,\ \boldsymbol{r}_i^j\boldsymbol{x} \leq \hat{\boldsymbol{b}}^j. \tag{32}$$

Since $\forall i,\ \boldsymbol{b}_i^j \leq \overline{\boldsymbol{b}^j} \wedge \overline{\boldsymbol{b}^j} = \hat{\boldsymbol{b}}^j$, we have that

$$\boldsymbol{x} \in P_i \Rightarrow \forall j,\ \boldsymbol{r}_i^j\boldsymbol{x} \leq \hat{\boldsymbol{b}}^j \Rightarrow \boldsymbol{x} \in \hat{P}_i \Rightarrow P_i \subseteq \hat{P}_i. \tag{33}$$

The above equation shows that $\hat{P}_i$ is an over-approximation of the polyhedron P_i obtained by relaxing the support functions to the upper limit of their intervals.

□

Using Theorem 4, we reduce the description of the Abstract Polyhedron to $[P] = \bigcup \hat{P}_i$.

The standard (non-interval) simplex algorithm visits a sequence of contiguous vertices $\boldsymbol{p}_i^p \in P_i$ and computes the *support function* of the last point $\boldsymbol{p}_i^n$ in this sequence in the direction of the objective function $\boldsymbol{v}$ (i.e., $\boldsymbol{v} \cdot \boldsymbol{p}_i^n$). Hence, to develop an interval simplex we need to describe the vertices of $[P]$ in a way that can be traversed and yields a solution $[\boldsymbol{v}] \cdot \mathbb{p}^n$.

Definition 4. *An extreme point of a polyhedron P is any point in the polyhedron touching at least one of its faces. Let $\boldsymbol{A}_i^p$ be a subset of rows of $\boldsymbol{A}_i$ with $\hat{\boldsymbol{b}}^p$ a subset of the corresponding rows in $\hat{\boldsymbol{b}}$ and $\boldsymbol{A}_i^{/p} \wedge \hat{\boldsymbol{b}}^{/p}$ their complementary rows and vector elements, respectively. An extreme point $\hat{\boldsymbol{p}}_i^p$ is defined by the equation:*

$$\boldsymbol{A}_i^p \hat{\boldsymbol{p}}_i^p = \hat{\boldsymbol{b}}^p \wedge \boldsymbol{A}_i^{/p} \hat{\boldsymbol{p}}_i^p \leq \hat{\boldsymbol{b}}^{/p}. \tag{34}$$

Definition 5. *A vertex of a polyhedron P is an extreme point in the polyhedron touching as many faces as the dimension of the polyhedron, namely*

$$\boldsymbol{A}_i^p \hat{\boldsymbol{p}}_i^p = \hat{\boldsymbol{b}}^p \wedge \boldsymbol{A}_i^{/p} \hat{\boldsymbol{p}}_i^p < \hat{\boldsymbol{b}}^{/p} : \hat{\boldsymbol{p}}_i^p \in \mathbb{R}^n \wedge |p| = n. \tag{35}$$

where $|p|$ represents the number of rows in $\boldsymbol{A}_i^p$.

Definition 6. *An abstract vertex $\mathbb{p}^p \in [P]$ is a hyperbox containing a corresponding vertex for each polyhedron in the collection $\hat{\boldsymbol{p}}_i^p \in \hat{P}_i$, so that $\mathbb{p}^p \supseteq Conv\left(\bigcup_i \hat{\boldsymbol{p}}_i^p\right)$. In the following we will replace the index p representing the basis of the vertex with the index k representing the order in which vertices are visited. For a visual representation, see Figure 1, where each set of halfplanes $\boldsymbol{r}_i^j \boldsymbol{x} \leq \hat{\boldsymbol{b}}^j$ (sets of lines marked $j = 1, 2, 3$ where each line represents the index i) intesects with another at an* abstract vertex *$\mathbb{p}^p$ (boxes). We can find multiple intersections inside each box corresponding to $\hat{\boldsymbol{p}}_i^p$.*

Definition 7. *A basis $\boldsymbol{B} \in \mathbb{R}^{n \times n}$ is a set of independent vectors, the linear combination of which spans the space $\mathbb{R}^n$.*

Theorem 5. *Given a pivot operation $pv\left(\boldsymbol{p}_i^k, \boldsymbol{p}_i^{k+1}\right) : \boldsymbol{p}_i^k \to \boldsymbol{p}_i^{k+1}$, an abstract pivot is a transformation*

$$pv\left(\mathbb{p}^k, \mathbb{p}^{k+1}\right) : \forall i,\ \hat{\boldsymbol{p}}_i^k \in \mathbb{p}^k \to \hat{\boldsymbol{p}}_i^{k+1} \in \mathbb{p}^{k+1}. \tag{36}$$

Notice that the pivot can be performed on any point, thus it is not limited to vertices or points within the polyhedron (this allows our abstract pivot *to take effect on all points in the hyperbox of the abstract vertex).*

Proof. Let $\boldsymbol{B}_i^k$ be a basis for $\hat{P}_i$ related to point $\boldsymbol{p}_i^k$, and such that

$$\boldsymbol{A}_i \boldsymbol{p}_i^k + \boldsymbol{B}_i^k \boldsymbol{s}_i^k = \hat{\boldsymbol{b}}, \tag{37}$$

where s_i^k is a set of auxiliary variables [4]. The pivot operation $pv\left(\boldsymbol{p}_i^k, \boldsymbol{p}_i^{k+1}\right)$ will change the basis such that $\boldsymbol{B}_i^{k+1} = \left(\boldsymbol{E}^k\right)^{-1} \boldsymbol{B}_i^k$. We therefore have

$$\begin{aligned}&\boldsymbol{A}_i \boldsymbol{p}_i^k + \boldsymbol{B}_i^k \boldsymbol{s}_i^k = \boldsymbol{A}_i \boldsymbol{p}_i^{k+1} + \boldsymbol{B}_i^{k+1} \boldsymbol{s}_i^{k+1} = \boldsymbol{A}_i \boldsymbol{p}_i^{k+1} + \left(\boldsymbol{E}^k\right)^{-1} \boldsymbol{B}_i^k \boldsymbol{s}_i^{k+1} \\ \Rightarrow &\boldsymbol{B}_i^k \boldsymbol{s}_i^k = \boldsymbol{A}_i \left(\boldsymbol{p}_i^{k+1} - \boldsymbol{p}_i^k\right) + \left(\boldsymbol{E}^k\right)^{-1} \boldsymbol{B}_i^k \boldsymbol{s}_i^{k+1}.\end{aligned} \tag{38}$$

The reason for using the inverse of $\boldsymbol{E}^k$ in the last equation is because implementations of the simplex often work on the inverse of the basis and the formula is not commutative using interval arithmetic. Since $\boldsymbol{E}^k$ creates a change between two bases spanning the same space, it is invertible.

Let $\mathbb{B}^k \supseteq \bigcup_i \boldsymbol{B}_i^k$ be an overapproximation of the basis related to $\mathbb{p}^k \in [P]$ (*i.e.* the set of bases relating to each point in $\mathbb{p}^k$). An abstract pivot preserves the over-approximation of the bases in equations (36) and (38) since:

$$\forall k, \exists \mathbb{E}^k \supseteq \bigcup_i \boldsymbol{E}_i^k : \mathbb{B}^{k+1} = \left(\mathbb{E}^k\right)^{-1} \mathbb{B}^k \supseteq \bigcup_i \boldsymbol{B}_i^{k+1}. \tag{39}$$

Applying interval arithmetic to equation (38) and moving $\mathbb{A}$ to the right, we obtain:

$$\begin{aligned} &\left(\mathbb{E}^k\right)^{-1} \mathbb{B}^k \mathbb{s}^{k+1} \supseteq \mathbb{A}\left(\mathbb{p}^k - \mathbb{p}^{k+1}\right) + \mathbb{B}^k \mathbb{s}^k \\ \Rightarrow &\forall i,\ \left(\mathbb{E}^k\right)^{-1} \boldsymbol{B}_i^k \boldsymbol{s}_i^{k+1} \supseteq \boldsymbol{A}_i\left(\boldsymbol{p}_i^k - \boldsymbol{p}_i^{k+1}\right) + \boldsymbol{B}_i^k \boldsymbol{s}_i^k. \end{aligned} \tag{40}$$

Equations (39) and (40) are satisfiable if we pick large enough intervals for the elements of $\mathbb{E}^k$, thus proving the theorem. □

A new problem arises regarding precision: whereas before we had disjoint vertices $\boldsymbol{p}^k \neq \boldsymbol{p}^{k+1}$, we now have possible intersections $\mathbb{p}^k \cap \mathbb{p}^{k+1} \neq \emptyset$. There are three consequences.

First, the over-approximation may become highly imprecise. Second, the algorithm may start cycling between the two intersecting vertices, which may cause the program to not terminate. While imprecision has been defined in Equation (16), the question is how to show completeness. We consider the definition of the vertices and Equation (18). If $\mathbb{A}\mathbb{p}^k$ is imprecise, then the base $\mathbb{B}^k$ is incomplete, and we abort the simplex, indicating that higher precision is required.

The third effect is that the corresponding confusion between two bases may cause the simplex to pivot erroneously on the second basis (i.e., once a vertex $\mathbb{p}^{k+1}$ is reached, the next pivot may start from $\mathbb{p}^j : \mathbb{p}^j \cap \mathbb{p}^{k+1} \neq \emptyset$ where $\exists i : \boldsymbol{A}_i \boldsymbol{p}_i^j + \boldsymbol{B}_i^{k+1} \boldsymbol{s}_i^{k+1} \neq \hat{\boldsymbol{b}}$). Therefore, before we pick a pivot, we must check that the current Abstract Basis matches the current Abstract Vertex: $\mathbb{A}\mathbb{p}^k + \mathbb{B}^k \mathbb{s}^k - \hat{\boldsymbol{b}} = 0$ (see Equation (18)). As with the other two cases, a failed check can be addressed by increasing the precision of the numerical algorithm. If the precision is not allowed to be increased indefinitely (i.e., it has a limit), then the procedure is not complete, since a number of problems (depending on the actual value of the precision) will not terminate with a valid result due to imprecision.

The final stage of the simplex, which corresponds to finding the support function $\max(\mathbb{v} \cdot \mathbb{p}^k)$ is trivially sound since it is the maximum of the resulting interval. Note that as stated at the beginning of this section, this is an over-approximation of the support function, given that $\mathbb{p}^k$ is an over-approximation in itself.

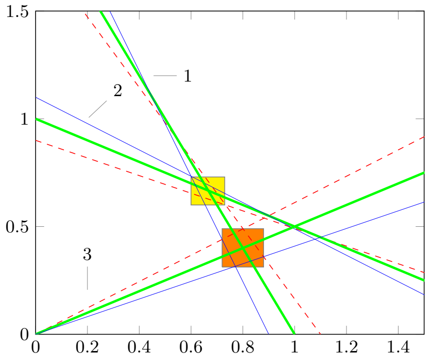

Fig. 1. Three interval half-planes with negative (dashed red), zero (thick green) and positive (thin blue) angular error representations. The yellow and orange areas (hypercubes) over-approximate all possible vertices of the resulting polyhedron at the given location. If these hypercubes partially intersect, the abstract vertex $\mathbb{p}^k$ must necessarily contain all intersecting hypercubes.

5.4 Vertex Enumeration

Vertex enumeration (step 9 in Algorithm 1) is an algorithm similar to simplex since it operates on the idea of visiting each vertex once in order to enumerate them.

The standard (non-interval) vertex enumeration algorithm starts by finding a base $\boldsymbol{V}^K$ which contains a number of vertices of the polyhedron. The index K is a set indicating the state of the algorithm by enumerating the rows of $\boldsymbol{A}$ that have been evaluated thus far. The process starts by pivoting over a number of different rows in $\boldsymbol{A}$ (using a simplex algorithm) and by selecting the feasible points visited, which are known to be vertices of the polyhedron. For this stage of the algorithm in the interval case, the use of a simplex as described in Section 5.3 ensures overall soundness. The base $\boldsymbol{V}^K$ is then iteratively expanded to $\boldsymbol{V}^{K+j}$ by exploring the j^{th} row of $\boldsymbol{A}$ (denoted $\boldsymbol{r}^j$). The corresponding pairs $(\boldsymbol{A}^{K+j}, \boldsymbol{V}^{K+j})$ are constructed using the information from $(\boldsymbol{A}^K, \boldsymbol{V}^K)$ as follows:

Theorem 6. *Let*

$$\boldsymbol{A}^K \in \mathbb{R}^{n_K \times p} \wedge \boldsymbol{r}_i^j \in \mathbb{R}^{1\times p} \wedge \boldsymbol{V}^K \in \mathbb{R}^{p\times m_K}.$$

where p is the dimension of the polyhedron, n_K the number of elements in the set K and m_K the number of vertices found up until stage K.

$$H_j{}^+ = \left\{\boldsymbol{x} : \boldsymbol{r}^j\boldsymbol{x} > 0\right\}, \quad H_j{}^- = \left\{\boldsymbol{x} : \boldsymbol{r}^j\boldsymbol{x} < 0\right\}, \quad H_j{}^0 = \left\{\boldsymbol{x} : \boldsymbol{r}^j\boldsymbol{x} = 0\right\} \tag{41}$$

be the spaces outside inside and on the j^{th} hyperplane with respect to the polyhedron. and

$$
\begin{aligned}
V^{K^+} &= \left\{ \boldsymbol{p}^+ \in \boldsymbol{V}^K \cap {H_j}^+ \right\} \\
V^{K^-} &= \left\{ \boldsymbol{p}^- \in \boldsymbol{V}^K \cap {H_j}^- \right\} \\
V^{K^0} &= \left\{ \boldsymbol{p}^0 \in \boldsymbol{V}^K \cap {H_j}^0 \right\}
\end{aligned} \tag{42}
$$

the existing vertex candidates lying in each of these spaces.
New vertex candidates are found as a linear combination of existing ones given a previously unused known constraint $\boldsymbol{r}^j$:

$$V^{K+j} = V^K \cup \left\{ \left(\boldsymbol{r}^j \boldsymbol{p}_k^+\right) \boldsymbol{p}_{k'}^- - \left(\boldsymbol{r}^j \boldsymbol{p}_{k'}^-\right) \boldsymbol{p}_k^+ : k \in [1 \; m_K^+] \wedge k' \in [1 \; m_K^-] \right\}. \tag{43}$$

where m_K^- and m_K^+ are the number of vertices in V^{K^-} and V^{K^+} respectively, $\boldsymbol{p}_k^-$ and $\boldsymbol{p}_k^+$ are points contained in the sets, and $\boldsymbol{r}^j$ is the selected row of $\boldsymbol{A}$ to be added to $\boldsymbol{A}^K$.

For the proof see [14].

Let us now consider the interval arithmetic equivalent of this theorem. Interval arithmetic ensures the soundness of the calculation

$$\mathbb{p}_{kk'} \in \boldsymbol{V}^{K+j} = \left(\mathbb{r}^j \mathbb{p}_k^+\right) \mathbb{p}_{k'}^- - \left(\mathbb{r}^j \mathbb{p}_{k'}^-\right) \mathbb{p}_k^+,$$

so all we need to ensure is the inclusions in (42).

If we expand for one of the sets, we get

$$[V^K]^+ = \left\{ \mathbb{p}^+ \in [\boldsymbol{V}^K] \cap [H_j]^+ : [H_j]^+ = \left\{ \mathbb{x} : \mathbb{r}^j \mathbb{x} > 0 \right\} \right\}$$

where the inclusion in $[H_j]^+$ becomes the concern, namely because using interval arithmetic we may find points that are either partially included (i.e., a portion of the interval, but not all of it, belongs to the set). Once again, we find that equation (18) ensures both the separation of the sets and the correctness of the inclusions.

Theorem 7. *Given the separation criteria in equation* (18)*, the sets* $[V^K]^+$*,* $[V^K]^-$ *and* $[V^K]^0$ *are disjoint.*

Proof. The proof is direct from the definitions. Any point that may intersect more than one set will be marked as imprecise by (18). □

As with the interval simplex, the algorithm is sound but may be incomplete. Since there always exists a precision that implies a sufficiently small rounding error (given that the error decreases monotonically with increasing precision), completeness can be achieved by increasing precision at a higher processing time cost.

Benchmark	Dimension	Unsound (ld)	Unsound (mp)	Sound (mpi)	exact
Building	48	$18s$	$185s$	$558s$	$t.o.$
issr10	10	$2s^*$	$23s$	$41s$	$t.o.$
Convoy Car 3	6	$0.3s$	$1.3s$	$3.6s$	$24.6s$
Convoy Car 2	3	$13ms$	$33ms$	$73ms$	$5.46s$
Parabola	4	$12ms$	$12ms$	$47ms$	$2.5s$

Table 1. Axelerator: time performance on various benchmarks. Dimension is the number of variables in the problem; ld denotes long double precision; mp is the required precision for the algorithm using non-interval arithmetic; mpi is the sound algorithm; exact is the sound algorithm run using exact arithmetic; t.o. denotes timeout. * returns invalid data (nan)

6 Experimental Results

The results discussed in the previous sections have been implemented in the tool Axelerator using *eigen* [17] for algebraic operations and *boost* [34] to manage intervals. Interval comparisons are implemented independently to follow our choice of ordering. The tool has been tested in a number of benchmarks (available with the tool) to determine the nature of the numerical errors. The benchmarks have been first run using unsound standard long double precision and multiple precision arithmetic with the required precision for the problem to be solved correctly (i.e., the precision demanded by our sound algorithm). The results are presented in Table 1.

It can be seen from the results that the cost of using sound arithmetic is approximately 3 times that of using floating points of the same precision. The bigger cost for larger dimensional models is the requirement to use a higher precision arithmetic. This happens because the intervals grow constantly (whereas regular floating point errors often cancel themselves out resulting in a smaller overall error), and the model requires the higher precision to maintain a representative model. Accepting larger errors however can result in both too conservative results and cycling in the simplex (which results in non-termination), so we must accept this need for the algorithm to work. The cost of using an exact arithmetic simplex to evaluate an interval Tableau is combinatorial, hence for example, a 10-dimensional Tableau would require 2^{10} operations which is clearly worse than any time increase required in this paper. The alternative, which is also requiring exact arithmetic in the eigen-decomposition, can be very costly (see last column in table 1). Thus, our algorithm offers a good tradeoff between fast unsound algorithms and slow exact ones.

7 Conclusion

We have developed a numerical multiple precision floating point interval algorithm for abstract acceleration. The results have shown that the round-of errors are relatively negligible for a large number of classes for a given precision. We have also demonstrated that the use of sound intervals comes at a relatively low processing cost of around 3x for the case of low precision systems (i.e., when the initially supplied precision suffices to ensure soundness), and a linear increase in cost with respect to precision when higher precision is required. Future work would include the use of variable precision arithmetic that would allow us to increase the precision only at the desired steps, *eg* when abstract vertices intersect.

References

1. Anderson, E., Bai, Z., Dongarra, J., Greenbaum, A., McKenney, A., Du Croz, J., Hammerling, S., Demmel, J., Bischof, C., Sorensen, D.: Lapack: A portable linear algebra library for high-performance computers. In: Proceedings of the 1990 ACM/IEEE conference on Supercomputing. pp. 2–11. IEEE Computer Society Press (1990)
2. Bak, S., Duggirala, P.S.: Hylaa: A tool for computing simulation-equivalent reachability for linear systems. In: Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control, HSCC 2017, Pittsburgh, PA, USA, April 18-20, 2017. pp. 173–178 (2017)
3. Bouissou, O., Mimram, S., Chapoutot, A.: Hyson: Set-based simulation of hybrid systems. In: Rapid System Prototyping (RSP), 2012 23rd IEEE International Symposium on. pp. 79–85. IEEE (2012)
4. Bradley, S., Hax, A., Magnanti, T.: Applied mathematical programming (1977)
5. Bustince, H., Galar, M., Bedregal, B., Kolesarova, A., Mesiar, R.: A new approach to interval-valued choquet integrals and the problem of ordering in interval-valued fuzzy set applications. IEEE Transactions on Fuzzy systems 21(6), 1150–1162 (2013)
6. Cattaruzza, D., Abate, A., Schrammel, P., Kroening, D.: Unbounded-time analysis of guarded LTI systems with inputs by abstract acceleration. In: SAS. LNCS, vol. 9291, pp. 312–331. Springer (2015)
7. Chen, L., Miné, A., Cousot, P.: A sound floating-point polyhedra abstract domain. In: Asian Symposium on Programming Languages and Systems. pp. 3–18. Springer (2008)
8. Chen, X., Ábrahám, E., Sankaranarayanan, S.: Flow*: An analyzer for non-linear hybrid systems. In: International Conference on Computer Aided Verification. pp. 258–263. Springer (2013)
9. Debreu, G.: Representation of a preference ordering by a numerical function. Decision processes 3, 159–165 (1954)
10. Felsner, S.: Interval orders: combinatorial structure and algorithms. Technische Universität Berlin (1992)
11. Fishburn, P.C.: Interval Orders and Interval Graphs – A Study of Partially Ordered Sets. Wiley (1985)
12. Fousse, L., Hanrot, G., Lefèvre, V., Pélissier, P., Zimmermann, P.: MPFR: A multiple-precision binary floating-point library with correct rounding. ACM Transactions on Mathematical Software (TOMS) 33(2), 13 (2007)
13. Frehse, G., Guernic, C.L., Donzé, A., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., Maler, O.: SpaceEx: Scalable verification of hybrid systems. In: CAV. LNCS, vol. 6806, pp. 379–395. Springer (2011)
14. Fukuda, K., Prodon, A.: Double description method revisited. In: Combinatorics and computer science, pp. 91–111. Springer (1996)

15. Gleixner, A.M., Steffy, D.E., Wolter, K.: Iterative refinement for linear programming. INFORMS Journal on Computing 28(3), 449–464 (2016)
16. Goldberg, D.: What every computer scientist should know about floating-point arithmetic. ACM Computing Surveys (CSUR) 23(1), 5–48 (1991)
17. Guennebaud, G., Jacob, B., et al.: Eigen v3. http://eigen.tuxfamily.org (2010)
18. Hong, Y., Pan, C.T.: A lower bound for the smallest singular value. Linear Algebra and its Applications 172, 27–32 (1992)
19. IEEE Task P754: IEEE 754-2008, Standard for Floating-Point Arithmetic (Aug 2008)
20. Jeannet, B., Miné, A.: Apron: A library of numerical abstract domains for static analysis. In: International Conference on Computer Aided Verification. pp. 661–667. Springer (2009)
21. Jeannet, B., Schrammel, P., Sankaranarayanan, S.: Abstract acceleration of general linear loops. In: POPL. pp. 529–540. ACM (2014)
22. Laub, A.J.: Matrix analysis for scientists and engineers. Siam (2005)
23. Lawson, C.L., Hanson, R.J., Kincaid, D.R., Krogh, F.T.: Basic linear algebra subprograms for Fortran usage. ACM Transactions on Mathematical Software (TOMS) 5(3), 308–323 (1979)
24. Le Lann, C., Boland, D., Constantinides, G.: The Krawczyk algorithm: Rigorous bounds for linear equation solution on an FPGA. In: International Symposium on Applied Reconfigurable Computing. pp. 287–295. Springer (2011)
25. MATLAB: version 7.10.0 (R2010a). The MathWorks Inc., Natick, Massachusetts (2010)
26. Monniaux, D.: On using floating-point computations to help an exact linear arithmetic decision procedure. In: International Conference on Computer Aided Verification. pp. 570–583. Springer (2009)
27. Muller, J.M., Brisebarre, N., De Dinechin, F., Jeannerod, C.P., Lefevre, V., Melquiond, G., Revol, N., Stehlé, D., Torres, S.: Handbook of floating-point arithmetic. Springer (2009)
28. Neumaier, A., Shcherbina, O.: Safe bounds in linear and mixed-integer linear programming. Mathematical Programming 99(2), 283–296 (2004)
29. de Oliveira, D.C.B., Monniaux, D.: Experiments on the feasibility of using a floating-point simplex in an SMT solver. In: PAAR@ IJCAR. pp. 19–28 (2012)
30. Pryce, J.: The forthcoming ieee standard 1788 for interval arithmetic. In: International Symposium on Scientific Computing, Computer Arithmetic, and Validated Numerics. pp. 23–39. Springer (2015)
31. Revol, N., Makino, K., Berz, M.: Taylor models and floating-point arithmetic: proof that arithmetic operations are validated in COSY. The Journal of Logic and Algebraic Programming 64(1), 135–154 (2005)
32. Rump, S.M.: Computational error bounds for multiple or nearly multiple eigenvalues. Linear Algebra and its Applications 324(1-3), 209–226 (2001)
33. Saad, Y.: Numerical methods for large eigenvalue problems, vol. 158. SIAM (1992)
34. Schäling, B.: The boost C++ libraries. Boris Schäling (2011)
35. Steele, J.M.: The Cauchy-Schwarz master class: an introduction to the art of mathematical inequalities. Cambridge University Press (2004)
36. Van Loan, C.F.: Matrix Computations. The Johns Hopkins University Press (1996)